File Transfer System in Java (TCP)

Documentation & Full Source Code

Overview
This project is a TCP-based file transfer system written in Java. It allows clients to connect to a server, list files in a shared directory, and download files with support for pause/resume. The server is multi-threaded, handling multiple clients concurrently.

Features

- Java Sockets (ServerSocket, Socket) for TCP networking
- Thread-per-client architecture for concurrency
- LIST command to view available files
- GET [offset] to download files with resume support
- Integrity verification via SHA-256 hashing (for small files)
- Client supports list, get, and get-all commands
- Human-readable text-based protocol

Architecture
The server listens on a specified TCP port (default 5050). Each client connection spawns a new thread (ClientHandler). The server shares files from the shared/ directory. The client saves files in the downloads/ directory, resuming from existing partial files when possible. Data is transferred in fixed 64KB chunks.

Protocol
Commands from client: LIST: Server responds with lines in the format name\tsize\tsha256|-, ending with a . line. GET <filename> [offset]: Server responds with OK <total_bytes> <sha256|-> and then streams file bytes from the specified offset.

Setup
1. Install Java 17+.
2. Place files to share in shared/ folder.
3. Compile code: javac FileTransferServer.java FileTransferClient.java
4. Run server: java FileTransferServer
5. Run client commands:
java FileTransferClient list
java FileTransferClient get filename.ext
java FileTransferClient get-all

Pause/Resume
The client checks the size of any existing partial file in the downloads directory. It uses this size as an offset when requesting the file from the server. The server seeks to this offset and streams the remaining bytes, enabling resume support. Users can interrupt the transfer and resume later.

Error Handling & Security
- The server validates file existence and offset values.
- The protocol is plaintext with no authentication or encryption. For production use, TLS and authentication should be added.
- Hashing is provided only for small files to balance performance and integrity checking.

Future Work
Enhancements could include file upload (PUT), parallel chunk transfers, GUI client, TLS encryption, user authentication, and resumable manifests for large file integrity verification.

FileTransferServer.java

```
import java.io.*;
import java.net.*;
import java.nio.file.*;
import java.security.*;
import java.util.*;

public class FileTransferServer {
    private static final int PORT = 5050;
    private static final String SHARED_DIR = "shared";
    private static final int CHUNK_SIZE = 64 * 1024; // 64 KB

    public static void main(String[] args) throws IOException {
        File dir = new File(SHARED_DIR);
        if (!dir.exists()) dir.mkdirs();

        ServerSocket serverSocket = new ServerSocket(PORT);
        System.out.println("[SERVER] Listening on port " + PORT);

        while (true) {
            Socket client = serverSocket.accept();
            System.out.println("[SERVER] Connection from " + client.getInetAddress());
            new Thread(new ClientHandler(client)).start();
        }
    }

    static class ClientHandler implements Runnable {
        private Socket socket;

        ClientHandler(Socket socket) {
            this.socket = socket;
        }

        @Override
        public void run() {
            try (
                BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            ) {
                out.write("HELLO FileTransferServer v1\n");
                out.flush();

                String line;
                while ((line = in.readLine()) != null) {
                    String[] parts = line.split(" ");
                    String cmd = parts[0].toUpperCase();

                    if (cmd.equals("LIST")) {
                        listFiles(out);
                    } else if (cmd.equals("GET")) {
                        if (parts.length < 2) {
                            out.write("ERR usage: GET <filename> [offset]\n");
                            out.flush();
                        } else {
                            String filename = parts[1];
                            long offset = (parts.length >= 3) ? Long.parseLong(parts[2]) : 0;
                            sendFile(out, socket.getOutputStream(), filename, offset);
                        }
                    } else {
                        out.write("ERR unknown command\n");
                        out.flush();
                    }
                }
            } catch (Exception e) {
                System.err.println("[SERVER] Error: " + e.getMessage());
            }
        }

        private void listFiles(BufferedWriter out) throws IOException {
            File dir = new File(SHARED_DIR);
            for (File file : Objects.requireNonNull(dir.listFiles())) {
                if (file.isFile()) {
                    long size = file.length();
                    String sha = (size <= 10 * 1024 * 1024) ? sha256(file) : "-";
                    out.write(file.getName() + "\t" + size + "\t" + sha + "\n");
                }
            }
            out.write(".\n");
```

```
            out.flush();
        }

        private void sendFile(BufferedWriter out, OutputStream rawOut, String filename, long offset) throws IOExcep
            File file = new File(SHARED_DIR, filename);
            if (!file.exists()) {
                out.write("ERR file not found\n");
                out.flush();
                return;
            }

            long total = file.length();
            if (offset < 0 || offset > total) {
                out.write("ERR invalid offset\n");
                out.flush();
                return;
            }

            String sha = (total <= 10 * 1024 * 1024) ? sha256(file) : "-";
            out.write("OK " + total + " " + sha + "\n");
            out.flush();

            try (RandomAccessFile raf = new RandomAccessFile(file, "r")) {
                raf.seek(offset);
                byte[] buffer = new byte[CHUNK_SIZE];
                int bytesRead;
                while ((bytesRead = raf.read(buffer)) != -1) {
                    rawOut.write(buffer, 0, bytesRead);
                    rawOut.flush();
                }
            }
        }

        private String sha256(File file) {
            try {
                MessageDigest digest = MessageDigest.getInstance("SHA-256");
                try (InputStream fis = new FileInputStream(file)) {
                    byte[] buffer = new byte[CHUNK_SIZE];
                    int n;
                    while ((n = fis.read(buffer)) > 0) {
                        digest.update(buffer, 0, n);
                    }
                }
                StringBuilder hex = new StringBuilder();
                for (byte b : digest.digest()) {
                    hex.append(String.format("%02x", b));
                }
                return hex.toString();
            } catch (Exception e) {
                return "-";
            }
        }
    }
}
```

FileTransferClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class FileTransferClient {
    private static final String HOST = "127.0.0.1";
    private static final int PORT = 5050;
    private static final String DOWNLOAD_DIR = "downloads";
    private static final int CHUNK_SIZE = 64 * 1024;

    public static void main(String[] args) throws IOException {
        if (args.length < 1) {
            usage();
            return;
        }

        String cmd = args[0];
        switch (cmd) {
            case "list" -> listRemote();
            case "get" -> {
                if (args.length < 2) usage();
                else getOne(args[1]);
            }
            case "get-all" -> {
                List<String> files = listRemote();
                for (String f : files) getOne(f);
            }
            default -> usage();
        }
    }

    private static void usage() {
        System.out.println(\"\"\"
            Usage:
              java FileTransferClient list
              java FileTransferClient get <filename>
              java FileTransferClient get-all
        \"\"\");
    }

    private static Socket connect() throws IOException {
        Socket s = new Socket(HOST, PORT);
        BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
        System.out.println("[SERVER] " + in.readLine());
        return s;
    }

    private static List<String> listRemote() throws IOException {
        Socket s = connect();
        List<String> files = new ArrayList<>();
        try (
            BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()))
        ) {
            out.write("LIST\n");
            out.flush();
            String line;
            System.out.printf("%-40s %12s %64s\n", "Name", "Bytes", "SHA256");
            while ((line = in.readLine()) != null) {
                if (line.equals(".")) break;
                String[] parts = line.split("\t");
                if (parts.length >= 2) {
                    System.out.printf("%-40s %12s %64s\n", parts[0], parts[1], parts.length >= 3 ? parts[2] : "-");
                    files.add(parts[0]);
                }
            }
        }
        s.close();
        return files;
    }

    private static void getOne(String filename) throws IOException {
        File dir = new File(DOWNLOAD_DIR);
        if (!dir.exists()) dir.mkdirs();
        File local = new File(dir, filename);

        long offset = local.exists() ? local.length() : 0;
```

```
        if (offset > 0) System.out.println("[RESUME] Resuming from " + offset);

        Socket s = connect();
        try (
            BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
            InputStream rawIn = s.getInputStream();
            RandomAccessFile raf = new RandomAccessFile(local, "rw")
        ) {
            out.write("GET " + filename + " " + offset + "\n");
            out.flush();

            String header = in.readLine();
            if (header == null || header.startsWith("ERR")) {
                System.out.println(header);
                return;
            }
            String[] parts = header.split(" ");
            long total = Long.parseLong(parts[1]);

            if (offset == total) {
                System.out.println("[SKIP] Already complete.");
                return;
            }

            raf.seek(offset);
            long received = offset;
            byte[] buffer = new byte[CHUNK_SIZE];
            int bytesRead;
            while ((bytesRead = rawIn.read(buffer)) != -1) {
                raf.write(buffer, 0, bytesRead);
                received += bytesRead;
                double pct = (received * 100.0) / total;
                System.out.printf("\r[PROGRESS] %d/%d bytes (%.2f%%)", received, total, pct);
            }
            System.out.println("\n[DONE] Download complete.");
        } finally {
            s.close();
        }
    }
}
```